

The Third Project Report

Samira Vaez Barenji

Student ID Number: 9921384

In this project, you can find a backtracking solution to the word break problem. The Dynamic Programming solution only finds whether it is possible to break a word or not. Here we need to print all possible word breaks.

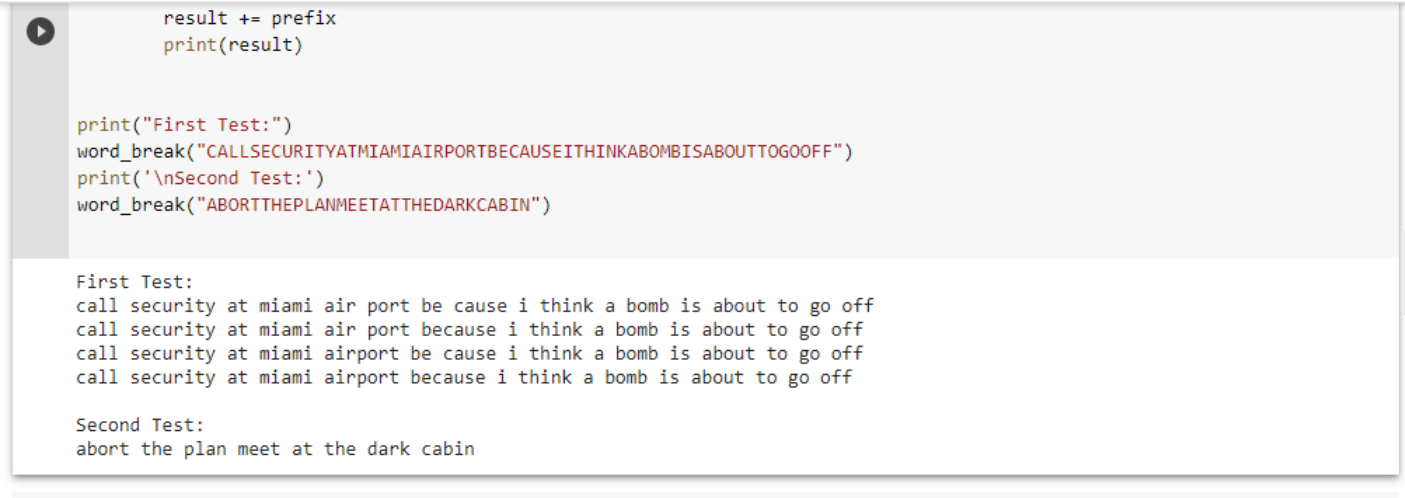
We start scanning the sentence from left. As we find a valid word, we need to check whether the rest of the sentence can make valid words or not. Because in some situations the first found word from the left side can leave a remaining portion which is not further separable. So, in that case, we should come back and leave the currently found word and keep on searching for the next word. And this process is recursive because to find out whether the right portion is separable or not, we need the same logic. So, we will use recursion and backtracking to solve this problem. To keep track of the found words we will use a stack. Whenever the right portion of the string does not make valid words, we pop the top string from the stack and continue finding.

We have listed a bunch of words in "vocab.txt" file as a dictionary to check whether a word is valid or not. We print all possible word breaks of a given string. In the *word_break* function, first, the input string is lowercased. Then, we process all prefixes one by one and extract substring in each iteration. If dictionary contains this prefix, then we check for remaining string. Otherwise, we ignore this prefix and try next. The *result* variable, stores the current prefix with spaces between words. So, the function takes an uppercased string with an empty variable called *result* as parameters and prints all possible word breaks in the output.

The worst-case time complexity of this algorithm is $O(2^n)$ because the resulting tree of the algorithm will have 2^n nodes at most. If we want to get all possible or valid sentences, we need to scroll through all the tree nodes, and the number of nodes never exceeds this value, where n is the length of the input string. The dictionary of words is stored in the *set* data structure of python which is

implemented as a hash table. So, we can expect to lookup/insert/delete in $O(1)$ on average.

The two sample sentences given in the project definition are given as input to the algorithm and you can see the resulting output image below.



```
result += prefix
print(result)

print("First Test:")
word_break("CALLSECURITYATMIAMIAIRPORTBECAUSEITHINKABOMBISABOUTTOGOOFF")
print('\nSecond Test:')
word_break("ABORTTHEPLANMEETATTHEDARKCABIN")
```

First Test:

call security at miami air port be cause i think a bomb is about to go off
call security at miami air port because i think a bomb is about to go off
call security at miami airport be cause i think a bomb is about to go off
call security at miami airport because i think a bomb is about to go off

Second Test:

abort the plan meet at the dark cabin